Topics Covered

1. Interactive Proof Systems
2. Graph Non-isomorphism
3. Interactive Proof for TAUTOLOGY

1 Interactive Proof Systems

On input $x$, an interactive proof system for a language $L$ involves a prover and verifier, both of which are given $x$. The prover can be thought of as a “Merlin”, meaning that there is no bound on its computational power. The verifier must run in polynomial time. If $x \in L$, the verifier accepts. If $x \notin L$, no dishonest prover can “trick” the verifier into accepting (we may relax this constraint later). For $L \in \text{NP}$, the proof system is:

```
Prover(x)  
If $x \in L$, find a witness $w$.  

Verifier(x)  
Run the verification algorithm $V$ on $x$ and $w$.  
Accept if $V$ accepts. Reject otherwise.
```

A pair of interactive TMs $P$ and $V$ is an interactive proof system for a language $L$ if:

- **Efficiency**: $V$’s runtime is polynomial in the size of its input.
- **Completeness**: For all $x \in L$, $P(x) \leftrightarrow V(x)$ makes $V(x)$ accept.
- **Soundness**: For all $x \notin L$, $\Pr[P(x) \leftrightarrow V(x) \text{ makes } V(x) \text{ accept}] \leq \frac{1}{2}$.

2 Graph Non-isomorphism

Graph non-isomorphism is a language that is not known to be in NP (it can be decided in quasi-polynomial or $O(n^{\text{polylog}(n)})$ time) but it is in coNP and has an interactive proof system. It is the language of pairs of graphs $(G_0, G_1)$ such that $G_0$ is not isomorphic to $G_1$. 
A graph \( G = (V_G, E_G) \) is \textbf{isomorphic} to \( H = (V_H, E_H) \) if there exists a permutation \( \pi : V_G \rightarrow V_H \) such that \( V_H = \{ \pi(v) \mid v \in V_G \} \) and \( E_H = \{ (\pi(u), \pi(v)) \mid (u, v) \in E_G \} \).

We write \( G \simeq H \).

Here is an interactive proof for graph non-isomorphism:

<table>
<thead>
<tr>
<th>Prover((G_0, G_1))</th>
<th>Verifier((G_0, G_1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select ( b \leftarrow {0,1} ).</td>
<td>Let ( H = \pi(G_b) ).</td>
</tr>
<tr>
<td>Select a random permutation ( \pi ).</td>
<td>Accept if ( b' = b ).</td>
</tr>
<tr>
<td>( b' = 0 ) if ( H \simeq G_0 ), ( \leftarrow H )</td>
<td>Otherwise, reject.</td>
</tr>
<tr>
<td>or ( b' = 1 ) if ( H \simeq G_1 ), ( \rightarrow b' )</td>
<td></td>
</tr>
</tbody>
</table>

The complexity class \( \text{IP} \) is the set of languages \( \{L \mid L \text{ has an interactive proof system}\} \). It turns out that \( \text{IP} = \text{PSPACE} \) (even though one only constrains time, and the other only constrains space!), but we won’t describe the proof today. It’s included in Sipser’s book.

3 Interactive Proof for Tautology

Another language in \( \text{IP} \) is \( \text{Tautology} = \{ \phi \mid \phi(a_1, \ldots, a_n) \text{ is true for all assignments } a_1, \ldots, a_n \text{ to the variables } x_1, \ldots, x_n \} \). To prove this, think of a 3CNF \( \phi = \phi_1 \land \cdots \land \phi_m \) as a polynomial:

- The literal \( x_1 \) has the equivalent polynomial \( p_{x_1} = x_1 \).
- A negated \( x_1 \) has the equivalent polynomial \( p_{\neg x_1} = (1 - x_1) \).
- The clause \( \phi_1 = x_1 \lor x_2 \lor x_3 \) corresponds to \( p_{\phi_1}(x_1, \ldots, x_n) = 1 - (1 - x_1)(1 - x_2)(1 - x_3) \). Note that this evaluates to 1 if the clause is satisfied, and 0 otherwise.
- The clause \( \phi_1 = x_1 \lor \neg x_2 \lor x_3 \) corresponds to \( p_{\phi_1}(x_1, \ldots, x_n) = 1 - (1 - x_1)(x_2)(1 - x_3) \).
- The formula \( \phi = \phi_1 \land \cdots \land \phi_m \) corresponds to \( p_{\phi}(x_1, \ldots, x_n) = \prod_{i=1}^m p_{\phi_i}(x_1, \ldots, x_n) \).

For any assignment \( a_1, \ldots, a_n \) to the variables \( x_1, \ldots, x_n \), \( p_{\phi}(a_1, \ldots, a_n) = 0 \) if \( \phi(a_1, \ldots, a_n) \) evaluates to false, and 1 if \( \phi(a_1, \ldots, a_n) \) evaluates to true. Thus, \( \phi \) is a tautology if and only if the sum of polynomial evaluations over all \( 2^n \) possible assignments is equal to \( 2^n \).

Define the following functions:

- \( f_0 = \sum_{a_1=0}^1 \sum_{a_2=0}^1 \cdots \sum_{a_n=0}^1 p_{\phi}(a_1, \ldots, a_n) = f_1(0) + f_1(1) \).
\[ f_1(z) = \sum_{a_1=0}^{1} \cdots \sum_{a_n=0}^{1} p_\phi(a_1 = z, \ldots, a_n) \]

\[ f_c(z_1, \ldots, z_i) = \sum_{a_{i+1}=0}^{1} \sum_{a_{i+2}=0}^{1} \cdots \sum_{a_n=0}^{1} p_\phi(a_1 = z_1, \ldots, a_i = z_i, a_{i+1}, \ldots, a_n) \]

Note that \( \phi \) is a tautology if and only if \( f_0 = 2^n \). Moreover, \( p_\phi \) is a polynomial of degree at most \( m \) for every variable \( x_i \). \( f_1(z) \) is a polynomial in one variable, with degree at most \( m \).

In general, \( f_{c-1}(z_1, \ldots, z_{c-1}) = f_c(z_1, \ldots, z_{c-1}, 0) + f_c(z_1, \ldots, z_{c-1}, 1) \). Here’s a first attempt at an interactive proof for Tautology:

<table>
<thead>
<tr>
<th>Prover(( \phi ))</th>
<th>Verifier(( \phi ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_0 = 2^n )</td>
<td>Prove ( f_0 ) correct.</td>
</tr>
<tr>
<td></td>
<td>( f_1 )</td>
</tr>
<tr>
<td></td>
<td>Prove ( f_1 ) correct.</td>
</tr>
<tr>
<td></td>
<td>( f_2 )</td>
</tr>
<tr>
<td></td>
<td>Check that ( f_2(1, 0) + f_2(1, 1) = f_1(1) ).</td>
</tr>
<tr>
<td></td>
<td>( \cdots )</td>
</tr>
<tr>
<td></td>
<td>( f_n = p_\phi )</td>
</tr>
</tbody>
</table>

Note that instead of sending polynomials, the prover can send points: \( f_0; f_1(0) \) and \( f_1(1) \); \( f_2(0, 0) \) and \( f_2(0, 1) \); and so on. This interactive proof system satisfies completeness, as all of the checks will pass if \( \phi \) is a tautology. It is also sound. If the prover sends some \( \hat{f}_0 \neq f_0 \), then it must lie on either \( f_1(0) \) or \( f_1(1) \). Informally, we claim that the lies must “percolate” and the verifier will eventually catch one. However, the runtime is a problem. To fix this, we need a way to compactly communicate the values and functions, and to perform the checks. Here’s another try:
Prover(\(\phi\)) | Verifier(\(\phi\))
---|---
\(f_0 = 2^n\) & Prove \(f_0\) correct. \\
\(g_1(z) = f_1(z)\) & Check that \(g_1(0) + g_1(1) = 2^n\). \\
\(r_1\) & Select \(r_1 \leftarrow \{0, 1\}\). \\
\(g_2(z) = f_2(r_1, z)\) & Check that \(g_2(0) + g_2(1) = f_1(r_1)\). \\
\(r_i\) & Select \(r_i \leftarrow \{0, 1\}\). \\
\(g_{i+1}(z) = f_{i+1}(r_1, \ldots, r_i, z)\) & Check that \(g_{i+1}(0) + g_{i+1}(1) = g_i(r_i)\). \\
\(r_n\) & Select \(r_n \leftarrow \{0, 1\}\). \\
\(g_{n-1}(z)\) & Check that \(g_{n-1}(r_n) = p_\phi(r_1, \ldots, r_n)\).

Once again, this proof system is complete; if \(\phi\) is a tautology, any random selection of points will yield valid polynomials. It is sound because the probability of selection a “bad point” is small. By a “bad point”, we mean points where a dishonest prover happens to satisfy the verifier’s equality tests. At the very end, the verifier compares \(g_{n-1}\) to ground truth (it knows \(p_\phi\) and can evaluate it on its own given an input. Here, the verifier will catch any false claims.